UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/681,556	10/08/2003	Henry Chang	100201439-1	7781

22879 7590 11/14/2007

HEWLETT PACKARD COMPANY
P O BOX 272400, 3404 E. HARMONY ROAD
INTELLECTUAL PROPERTY ADMINISTRATION
FORT COLLINS, CO 80527-2400

EXAMINER	
WANG, BEN C	
ART UNIT	**PAPER NUMBER**
2192	

MAIL DATE	DELIVERY MODE
11/14/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/681,556	CHANG ET AL.	
	Examiner	Art Unit	
	Ben C. Wang	2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *05 September 2007*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-2 and 4-17* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-2 and 4-17* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. Applicant's amendment dated September 5, 2007, responding to the June 5, 2007 Office action provided in the rejection of claims 1-2 and 4-17.

Claims 1-2 and 4-17remain pending in the application and which have been fully considered by the examiner.

Status of claim 3 should be --cancelled--. instead of "Original" listed on page 3 (see REMARKS, on page 10, 1st Par.)

Claim Rejections – 35 USC § 103(a)

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

2. Claims 1, 4, 5, 8-12, 15 and 17 are rejected under 35 U.S.C. 103(a) as being unpatentable over Shmuel Ur et al., (US 2003/0110474 A1) (hereinafter 'Shmuel Ur') in view of Cahill et al., *'The Java Metrics Reporter – An Extensible Tool for OO Software Analysis'*, 2002 IEEE (hereinafter 'Cahill'), Benlarbi et al., *'Polymorphism Measures for Early Risk Prediction'*, 1999, ACM) (hereinafter 'Benlarbi') and further in view of Mitchell et al., '*Toward a definition of run-time object-oriented metrics*", July 22nd, 2003, ECOOP (hereinafter 'Mitchell')

3. **As to claim 1** (Previously Presented), Shmuel Ur discloses that a method for testing software, comprising: examining an application software program including calls to both a static analysis tool and a dynamic analysis tool (e.g., [0030], lines 1-7); testing said system classes (the dominating blocks) in order according to said corresponding proportional weighing attributes (e.g., [0005], lines 12-15; [0006], lines 22-26 and [0007]).

But Shmuel Ur does not explicitly disclose calls to system classes from the examining; determining a static use count of said system classes from the examining; deriving a dynamic use count of each of said system classes during operation of said application software program from the examining; assigning a proportional weighing attribute to each system class based on its corresponding static use count and dynamic use count

However, in an analogous art, Cahill discloses calls to system classes form the examining (e.g., Sec. 2.3, the table on page 512, the 1st entry – System, Root Classes); determining a static use count of said system classes from the examining (e.g., Sec. 3.1 Present Selection and Rationale, 1st paragraph – fifteen metrics organized into the categories Basic (Sec. 3.2), Complexity (3.3), Inheritance (Sec. 3.4) and Polymorphism (Sec.3.5). Sec. 3.4, Method Inheritance Factor (MIF), lines 1-5 for static use count portion; Sec. 3.5, Polymorphism Factor (PF), lines 5-11 for dynamic/static use count portion).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of both Shmuel Ur and Cahill in

order to calls to system classes from the examining; determining a static use count of said system classes from the examining in Shmuel Ur's system.

The motivation is to put highly used software into system classes that can be shared between various application classes without mixing them with application code.

It is also well know in the art that polymorphism metrics measure includes both static and dynamic information (e.g., see Benlarbi, Sec. 3.1 Forms of Polymorphism in OO design, 5th paragraph – classification of polymorphic behaviors includes polymorphic behaviors that are based on run-time binding (dynamic) decisions as well as on compile time (static) linking decision).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of Shmuel Ur, Cahill, and Benlarbi in order to collect both static and dynamic use count and assigning proportional weighing attributes.

The motivation is that inheritance factor metric (for static use count) is useful in providing a direct measure of inheritance in a system, and from this, the level or reuse in the system, and polymorphism factor metric (for dynamic count use) is useful in indicating complexity, comprehensibility and maintainability.

Although Cahill discloses both static and dynamic counts (e.g., Sec. 3.1 – Sec. 3.5), but Shmuel UR, Cahill, and Benlarbi do not explicitly disclose deriving a dynamic use count of each of said system classes during operation of said application software program from the examining; assigning a proportional weighing attribute to each system class based on its corresponding static use count and dynamic use count.

However, in an analogous art of toward a definition of run-time object-oriented metrics, Mitchell discloses deriving a dynamic use count of each of said system classes during operation of said application software program from the examining; assigning a proportional weighing attribute to each system class based on its corresponding static use count and dynamic use count (e.g., Sec. 1, 3rd Para. – static metrics alone may be insufficient in evaluating the dynamic behavior of an application at run time, as its behavior will be influenced by the operational environment as well as the complexity of the source code; Sec. 2, 2nd Para. – to quantify the external quality of a software product using a set of dynamic metrics calculated at run-time that evaluate the product's complexity. These metrics by themselves can provide us with useful information, and can also help to calibrate the information obtained from a static analysis, 5th Para. – The quality of a software product will be influenced by its operational environment as well as the source code complexity, therefore it was believed that measures that assess run-time quality may aid in the analysis of software quality. Static coupling and cohesion metrics deal with the architectural aspects of a software system, whereas run-time measures necessarily deal also with the behavioral aspects of the system; Sec. III Work Done To Date, 1st Para., 3rd Para., 4th Para., 5th Para. – it is known that a one-to-one relationship does not exist between static and dynamic metrics but our study indicated that there might be some co-relation; Sec. A. Relationship with Software Testing, 4th Para. – to determine if a quantifiable correlation exists between the information obtained from a static and dynamic analysis of a program; clearly, a static analysis is relatively independent of program behavior, whereas any run-time analysis will be fundamentally

influenced by the testing strategy and test inputs; Fig. 1 – Degree of Static and Dynamic Coupling within a group of classes for the non-API classes of the programs from SPEC JVM98 benchmark suite; Sec. A – Coupling Metrics; Sec. B – Cohesion Metrics – Figs 2-3 give a pictorial representation of the relationship between the static and dynamic cohesion data ...).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of Mitchell into Shmuel Ur-Cahill-Benlarbi system to further derive a dynamic use count of each of said system classes during operation of said application software program from the examining; assign a proportional weighing attribute to each system class based on its corresponding static use count and dynamic use count in Shmuel Ur-Cahill-Benlarbi system.

The motivation is that it would further enhance the Shmuel Ur-Cahill-Benlarbi's system by taking, advancing and/or incorporating Mitchell's system which offers significant advantages to quantify the external quality of a software product using a set of dynamic metrics calculated at run-time that evaluate the product's complexity; these metrics by themselves can provide us with useful information, and can also help to calibrate the information obtained from a static analysis as once suggested by Mitchell (e.g., Sec. II. – Related Work, 2nd Para.).

4. **As to claim 4** (original) (incorporating the rejection in claim 1), Shmuel Ur does not specifically disclose that producing a static use count further comprises assigning a

static observation percentage to each system class by dividing said static use count by a sum of all static use counts.

However, in an analogous art, Cahill discloses that producing a static use count further comprises assigning a static observation percentage to each system class by dividing said static use count by a sum of all static use counts (e.g., Sec. 3.4, Method Inheritance Factor (MIF), lines 2-5).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of both Shmuel Ur and Cahill in order to produce static observation percentage for each system class.

The motivation is that static observation percentage is useful in providing a direct measure of inheritance in a system, and from this, the level of reuse in the system.

5. **As to claim 5** (original) (incorporating the rejection in claim 1), Shmuel Ur does not specifically disclose that producing a dynamic use count further comprises assigning a dynamic observation percentage to each system class by dividing said dynamic use count by a sum of all dynamic use counts.

However, in an analogous art, Cahill discloses that producing a dynamic use count further comprises assigning a dynamic observation percentage to each system class by dividing said dynamic use count by a sum of all dynamic use counts (e.g., Sec. 3.5, Polymorphism Factor (PF), Lines 2-7).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of both Shmuel Ur and Cahill in

order to produce dynamic observation percentage for each system class. The motivation is that dynamic observation percentage is useful in indicating complexity, comprehensibility and maintainability.

6. **As to claim 8** (original) (incorporating the rejection in claim 1), Shmuel Ur discloses that the testing system classes further comprises ending a test when a testing resource is exhausted and prior to testing a last entry have a least weight (e.g., [0004], lines 12-14).

7. **As to claim 9** (original) (incorporating the rejection in claim 8), Shmuel Ur discloses that the testing system classes further comprises ending a test when at least a limit of available time or funding is exhausted and prior to testing a last entry having a least weight (e.g., [0004], lines 12-14).

8. **As to claim 10** (Previously Presented), Shmuel Ur discloses a machine-readable medium on which is encoded machine readable code for testing object-oriented system software having system classes, the machine readable code comprising: machine-readable code for running a static analysis tool for examining an application software program; machine-readable code for running a dynamic analysis tool for examining the application software program (e.g., [0030], lines 1-7); machine-readable code for testing said system classes (the dominating blocks) in order, based on the assigned weight,

from a first entry having a greatest weight (e.g., [0005], lines 12-15; [0006], lines 22-26 and [0007]).

But, Shmuel Ur does not explicitly disclose calls to system classes. However, in an analogous art, Cahill discloses calls to system classes (e.g., Sec. 2.3, the table on page 512, the 1st entry – System, Root Classes).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of both Shmuel Ur and Cahill in order to call to system classes.

The motivation is to put highly used software into system classes that can be shared between various application classes without mixing them with application code.

Further, Shmuel Ur does not disclose that machine-readable code for determining a static use count of the system classes in the application software program from the result; machine-readable code for assigning to each system class a weight based on the static use count and the dynamic use count.

However, in an analogous art, Cahill discloses that machine-readable code for determining a static use count of the system classes in the application software program from the result (e.g., Sec. 3.1 Present Selection and Rationale, 1st paragraph – fifteen metrics organized into the categories Basic (Sec. 3.2), Complexity (3.3), Inheritance (Sec. 3.4) and Polymorphism (Sec.3.5). Sec. 3.4, Method Inheritance Factor (MIF), lines 1-5 for static use count portion; Sec. 3.5, Polymorphism Factor (PF), lines 5-11 for dynamic/static use count portion).

It is also well know in the art that polymorphism metrics measure includes both static and dynamic information (e.g., see Benlarbi, Sec. 3.1 Forms of Polymorphism in OO design, 5th paragraph – classification of polymorphic behaviors includes polymorphic behaviors that are based on run-time binding (dynamic) decisions as well as on compile time (static) linking decision).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of Shmuel Ur, Cahill and Benlarbi in order to determine a static use count of the system classes in the application software program from the result in Shmuel Ur's system.

The motivation is that inheritance factor metric (for static use count) is useful in providing a direct measure of inheritance in a system, and from this, the level or reuse in the system, and polymorphism factor metric (for dynamic count use) is useful in indicating complexity, comprehensibility and maintainability. Shmuel Ur discloses that testing said 'system classes' (the dominating blocks) in order according to said corresponding proportional weighing attributes (e.g., [0005], lines 12-15; [0006], lines 22-26 and [0007]).

Although Cahill discloses both static and dynamic counts (e.g., Sec. 3.1 – Sec. 3.5), but Shmuel UR, Cahill, and Benlarbi do not explicitly disclose machine-readable code for producing a dynamic use count based on the application software program's dynamic use of the system functions while running the application software program and assigning to each system class a weight based on the static use count and the dynamic use count.

However, in an analogous art of toward a definition of run-time object-oriented metrics, Mitchell discloses machine-readable code for producing a dynamic use count based on the application software program's dynamic use of the system functions while running the application software program and assigning to each system class a weight based the static use count and the dynamic use count (e.g., Sec. 1, 3rd Para. – static metrics alone may be insufficient in evaluating the dynamic behavior of an application at run time, as its behavior will be influenced by the operational environment as well as the complexity of the source code; Sec. 2, 2nd Para. – to quantify the external quality of a software product using a set of dynamic metrics calculated at run-time that evaluate the product's complexity. These metrics by themselves can provide us with useful information, and can also help to calibrate the information obtained from a static analysis, 5th Para. – the quality of a software product will be influenced by its operational environment as well as the source code complexity, therefore it was believed that measures that assess run-time quality may aid in the analysis of software quality. Static coupling and cohesion metrics deal with the architectural aspects of a software system, whereas run-time measures necessarily deal also with the behavioral aspects of the system; Sec. III Work Done To Date, 1st Para., 3rd Para., 4th Para., 5th Para. – it is known that a one-to-one relationship does not exist between static and dynamic metrics but our study indicated that there might be some co-relation; Sec. A. Relationship with Software Testing, 4th Para. – to determine if a quantifiable correlation exists between the information obtained from a static and dynamic analysis of a program; clearly, a static analysis is relatively independent of program behavior, whereas any run-time

analysis will be fundamentally influenced by the testing strategy and test inputs; Fig. 1 – Degree of Static and Dynamic Coupling within a group of classes for the non-API classes of the programs from SPEC JVM98 benchmark suite; Sec. A – Coupling Metrics; Sec. B – Cohesion Metrics – Figs 2-3 give a pictorial representation of the relationship between the static and dynamic cohesion data ...).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of Mitchell into Shmuel Ur-Cahill-Benlarbi system to further provide machine-readable code for assigning to each system class a weight based the static use count and the dynamic use count in Shmuel Ur-Cahill-Benlarbi system.

The motivation is that it would further enhance the Shmuel Ur-Cahill-Benlarbi's system by taking, advancing and/or incorporating Mitchell's system which offers significant advantages to quantify the external quality of a software product using a set of dynamic metrics calculated at run-time that evaluate the product's complexity; these metrics by themselves can provide us with useful information, and can also help to calibrate the information obtained from a static analysis as once suggested by Mitchell (Sec. II. – Related Work, 2nd Para.).

9. **As to Claim 11** (Original) (incorporating the rejection in claim 10), please refer to claim **4** as set forth accordingly.

10. **As to claim 12** (Original) (incorporating the rejection in claim 10), please refer to claim **5** as set forth accordingly.

11. **As to claim 15** (Previously Presented), Shmuel Ur discloses a machine-readable medium on which is encoded a software tester program code, the software tester program code comprising: means for examining an application software program including calls to both a static analysis tool and a dynamic analysis tool (e.g., [0030], lines 1-7); means for testing said system classes (the dominating blocks) in order according to said corresponding proportional weighing attributes (e.g., [0005], lines 12-15; [0006], lines 22-26 and [0007]).

But, Shmuel Ur does not explicitly disclose to call to system classes.

However, in the analogous art, Cahill discloses to call to system classes (e.g., Sec. 2.3, the table on page 512, the 1st entry – System, Root Classes).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of both Shmuel Ur and Cahill in order to call to system classes.

The motivation is to put highly used software into system classes that can be shared between various application classes without mixing them with application code.

Further, Shmuel Ur does not disclose means for determining a static use count of said system classes from the examining; means for deriving a dynamic use count of each of said system classes during operation of said application software program from

the examining; means for assigning a proportional weighing attribute to each system class based on its corresponding static use count and dynamic use count.

However, in an analogous art, Cahill discloses that means for determining a static use count of said system classes from the examining (e.g., Sec. 3.1 Present Selection and Rationale, 1st paragraph – fifteen metrics organized into the categories Basic (Sec. 3.2), Complexity (3.3), Inheritance (Sec. 3.4) and Polymorphism (Sec.3.5). Sec. 3.4, Method Inheritance Factor (MIF), lines 1-5 for static use count portion; Sec. 3.5, Polymorphism Factor (PF), lines 5-11 for dynamic/static use count portion).

It is also well know in the art that polymorphism metrics measure includes both static and dynamic information (e.g., see Benlarbi, Sec. 3.1 Forms of Polymorphism in OO design, 5th paragraph – classification of polymorphic behaviors includes polymorphic behaviors that are based on run-time binding (dynamic) decisions as well as on compile time (static) linking decision).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of Shmuel Ur, Cahill and Benlarbi in order to collect both static and dynamic use count and assigning proportional weighing attributes.

The motivation is that inheritance factor metric (for static use count) is useful in providing a direct measure of inheritance in a system, and from this, the level or reuse in the system, and polymorphism factor metric (for dynamic count use) is useful in indicating complexity, comprehensibility and maintainability.

Although Cahill discloses both static and dynamic counts (e.g., Sec. 3.1 – Sec. 3.5), but Shmuel UR, Cahill, and Benlarbi do not explicitly disclose means for deriving a dynamic use count of each of said system classes during operation of said application software program from the examining; means for assigning a proportional weighing attribute to each system class based on its corresponding static use count and dynamic use count.

However, in an analogous art of toward a definition of run-time object-oriented metrics, Mitchell discloses means for means for deriving a dynamic use count of each of said system classes during operation of said application software program from the examining; assigning a proportional weighing attribute to each system class based on its corresponding static use count and dynamic use count (e.g., Sec. 1, 3rd Para. – static metrics alone may be insufficient in evaluating the dynamic behavior of an application at run time, as its behavior will be influenced by the operational environment as well as the complexity of the source code; Sec. 2, 2nd Para. – to quantify the external quality of a software product using a set of dynamic metrics calculated at run-time that evaluate the product's complexity. These metrics by themselves can provide us with useful information, and can also help to calibrate the information obtained from a static analysis, 5th Para. – the quality of a software product will be influenced by its operational environment as well as the source code complexity, therefore it was believed that measures that assess run-time quality may aid in the analysis of software quality. Static coupling and cohesion metrics deal with the architectural aspects of a software system, whereas run-time measures necessarily deal also with the behavioral aspects of the

system; Sec. III Work Done To Date, 1st Para., 3rd Para., 4th Para., 5th Para. – it is known that a one-to-one relationship does not exist between static and dynamic metrics but our study indicated that there might be some co-relation; Sec. A. Relationship with Software Testing, 4th Para. – to determine if a quantifiable correlation exists between the information obtained from a static and dynamic analysis of a program; clearly, a static analysis is relatively independent of program behavior, whereas any run-time analysis will be fundamentally influenced by the testing strategy and test inputs; Fig. 1 – Degree of Static and Dynamic Coupling within a group of classes for the non-API classes of the programs from SPEC JVM98 benchmark suite; Sec. A – Coupling Metrics; Sec. B – Cohesion Metrics – Figs 2-3 give a pictorial representation of the relationship between the static and dynamic cohesion data ...).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of Mitchell into Shmuel Ur-Cahill-Benlarbi system to further provide means for deriving a dynamic use count of each of said system classes during operation of said application software program from the examining; means for assigning a proportional weighing attribute to each system class based on its corresponding static use count and dynamic use count in Shmuel Ur-Cahill-Benlarbi system.

The motivation is that it would further enhance the Shmuel Ur-Cahill-Benlarbi's system by taking, advancing and/or incorporating Mitchell's system which offers significant advantages to quantify the external quality of a software product using a set of dynamic metrics calculated at run-time that evaluate the product's complexity; these

metrics by themselves can provide us with useful information, and can also help to calibrate the information obtained from a static analysis as once suggested by Mitchell (e.g., Sec. II. – Related Work, 2nd Para.).

12. **As to claim 17** (Previously Presented), Shmuel Ur discloses a business model for testing software, comprising: setting a resource limit on the available time or money that is devoted to testing a particular application software program; examining an application software program including calls to both a static analysis tool and a dynamic analysis tool (e.g., [0030], lines 1-7); stopping testing when said resource limit is reached ([0097], lines 1-6; [0049]); testing said 'system classes' (the dominating blocks) in order according to said corresponding proportional weighing attributes (e.g., [0005], lines 12-15; [0006], lines 22-26 and [0007]); a business model factors for testing software ([0004], lines 12-14).

But Shmuel Ur does not explicitly disclose to call to system classes. However, in the analogous art, Cahill discloses to call to system classes (e.g., Sec. 2.3, the table on page 512, the 1st entry – System, Root Classes).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of both Shmuel Ur and Cahill in order to call to system classes.

The motivation is to put highly used software into system classes that can be shared between various application classes without mixing them with application code.

Shmuel Ur does not disclose that determining a static use count of said system classes from the examining; deriving a dynamic use count of each of said system classes during operation of said application software program from the examining; assigning a proportional weighing attribute to each system class based on its corresponding static use count and dynamic use count.

However, in an analogous art, Cahill discloses that determining a static use count of said system classes from the examining; deriving a dynamic use count of each of said system classes during operation of said application software program from the examining (e.g., Sec. 3.1 Present Selection and Rationale, 1st paragraph – fifteen metrics organized into the categories Basic (Sec. 3.2), Complexity (3.3), Inheritance (Sec. 3.4) and Polymorphism (Sec.3.5). Sec. 3.4, Method Inheritance Factor (MIF), lines 1-5 for static use count portion; Sec. 3.5, Polymorphism Factor (PF), lines 5-11 for dynamic/static use count portion).

It is also well know in the art that polymorphism metrics measure includes both static and dynamic information (e.g., see Benlarbi, Sec. 3.1 Forms of Polymorphism in OO design, 5th paragraph – classification of polymorphic behaviors includes polymorphic behaviors that are based on run-time binding (dynamic) decisions as well as on compile time (static) linking decision).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of Shmuel Ur, Cahill and Benlarbi in order to collect both static and dynamic use count and assigning proportional weighing attributes.

The motivation is that inheritance factor metric (for static use count) is useful in providing a direct measure of inheritance in a system, and from this, the level or reuse in the system, and polymorphism factor metric (for dynamic count use) is useful in indicating complexity, comprehensibility and maintainability.

Although Cahill discloses both static and dynamic counts (e.g., Sec. 3.1 – Sec. 3.5), but Shmuel UR, Cahill, and Benlarbi do not explicitly disclose assigning a proportional weighing attribute to each system class based on its corresponding static use count and dynamic use count.

However, in an analogous art of toward a definition of run-time object-oriented metrics, Mitchell discloses assigning a proportional weighing attribute to each system class based on its corresponding static use count and dynamic use count (e.g., Sec. 1, 3rd Para. – static metrics alone may be insufficient in evaluating the dynamic behavior of an application at run time, as its behavior will be influenced by the operational environment as well as the complexity of the source code; Sec. 2, 2nd Para. – to quantify the external quality of a software product using a set of dynamic metrics calculated at run-time that evaluate the product's complexity. These metrics by themselves can provide us with useful information, and can also help to calibrate the information obtained from a static analysis, 5th Para. – the quality of a software product will be influenced by its operational environment as well as the source code complexity, therefore it was believed that measures that assess run-time quality may aid in the analysis of software quality. Static coupling and cohesion metrics deal with the architectural aspects of a software system, whereas run-time measures necessarily

deal also with the behavioral aspects of the system; Sec. III Work Done To Date, 1st Para., 3rd Para., 4th Para., 5th Para. – it is known that a one-to-one relationship does not exist between static and dynamic metrics but our study indicated that there might be some co-relation; Sec. A. Relationship with Software Testing, 4th Para. – to determine if a quantifiable correlation exists between the information obtained from a static and dynamic analysis of a program; clearly, a static analysis is relatively independent of program behavior, whereas any run-time analysis will be fundamentally influenced by the testing strategy and test inputs; Fig. 1 – Degree of Static and Dynamic Coupling within a group of classes for the non-API classes of the programs from SPEC JVM98 benchmark suite; Sec. A – Coupling Metrics; Sec. B – Cohesion Metrics – Figs 2-3 give a pictorial representation of the relationship between the static and dynamic cohesion data ...).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of Mitchell into Shmuel Ur-Cahill-Benlarbi system to further assign a proportional weighing attribute to each system class based on its corresponding static use count and dynamic use count in Shmuel Ur-Cahill-Benlarbi system.

The motivation is that it would further enhance the Shmuel Ur-Cahill-Benlarbi's system by taking, advancing and/or incorporating Mitchell's system which offers significant advantages to quantify the external quality of a software product using a set of dynamic metrics calculated at run-time that evaluate the product's complexity; these metrics by themselves can provide us with useful information, and can also help to

calibrate the information obtained from a static analysis as once suggested by Mitchell (e.g., Sec. II. – Related Work, 2nd Para.).

13. Claims 6 and 13 are rejected under 35 U.S.C. 103(a) as being unpatentable over Shmuel Ur, in view of Cahill, Benlarbi, and Mitchell and further in view of T. Ball (*'The Concept of Dynamic Analysis'*, Bell Laboratories Lucent Technologies, Oct, 1999') (hereinafter 'Ball')

14. **As to claim 6** (original) (incorporating the rejection in claim 1), Shmuel Ur, Cahill, Benlarbi, and Mitchell do not disclose that producing a static use count further comprises assigning a static observation percentage to each system class by diving the static use count by a sum of all static use counts; and producing a dynamic use count further comprises assigning a dynamic observation percentage to each system class by dividing the dynamic use count by a sum of all dynamic use counts.

However, in an analogous art, Ball discloses that producing a static use count further comprises assigning a static observation percentage to each system class by diving the static use count by a sum of all static use counts; and producing a dynamic use count further comprises assigning a dynamic observation percentage to each system class by dividing the dynamic use count by a sum of all dynamic use counts (e.g., Sec. 1, dynamic and static analyses are complementary techniques in a number of dimensions: a) completeness, b) scope, and c) precision).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of Shmuel Ur, Cahill, Benlarbi, Mitchell and Ball in order to produce static plus dynamic observation percentages for each system class.

The motivation is that both static and dynamic observation percentages for each system class are complementary metrics factors to product more complete, precise, scope combined software metrics report.

15. **As to claim 13** (Original) (incorporating the rejection in claim 10), please refer to claim **6** as set forth accordingly.

16. Claims 2 and 16 are rejected under 35 U.S.C. 103(a) as being unpatentable over Shmuel Ur, Cahill, Benlarbi, and Mitchell in view of Kuzmin et al., (pat. no. US 7,039,902 B2) (hereinafter 'Kuzmin'),

17. **As to claim 2** (original) (incorporating the rejection in claim 1), Shmuel Ur, Cahill, Benlarbi, and Mitchell do not disclose that wherein the step of testing is such that only the most heavily weighted portion of all such system classes are test at all.

However, in an analogous art, Kuzmin discloses that wherein the step of testing is such that only the most heavily weighted portion of all such system classes are tested at all (e.g., Col. 5, lines 48, 53).

It would have been obvious to one of ordinary skill in the art at the time of invention was made to combine the teachings of Shmuel Ur, Cahill, Benlarbi, Mitchell and Kuzmin in order to focus on testing in most heavily weighted portion of all such system classes.

The motivation is that prioritization of untested portions of code is advantageous because it finds the untested portions with the most potential impact in the body of application code.

18. **As to claim 16** (Original) (incorporating the rejection in claim 15), please refer to claim **2** as set forth accordingly.

Allowable Subject Matter

19. Claims 7 and 14 are objected to as being dependent upon a rejected base claim, but would be allowable if rewritten to overcome all the limitations of the base claim and any intervening claims.

The following is an examiner's statement of reasons for allowance:

Regarding claims 7 and 14, prior art of record fails to reasonably show or suggest the specific weighting scheme as claimed. Specifically, the assigning a first weight defined by a first constant plus a sum of the static use count plus the dynamic use count, a second weight equal to the first constant, further assigning a third weight as a second constant that is less than the first constant (which is added a sum of the

static and dynamic observation percentage), and assigning to all remaining classes a fourth weight less than the second constant.

Response to Arguments

20. Applicant's arguments filed on September 5, 2007 have been fully considered but they are not persuasive.

In the remarks, Applicant argues that:

a) Cahill et al. provides not discussion regarding a reliance on a determined static use count of system classes and derived dynamic use count of each of the system classes in order to assign a proportional weighting attribute to each system class (see REMARKS on P. 15, 2nd Par.)

b) Claim 6 depends on claim 1, and claim 13 depends on 10. However, the Office Action used Mitchell et al. to reject claims 1 and 10 but not reject claims 6 and 13 (see REMARKS on P. 17, Sec. of 'Claims 6 and 10')

c) Claim 2 depends on claim 1, and claim 16 depends on 15. However, the Office Action used Mitchell et al. to reject claims 1 and 15 but not reject claims 2 and 6 (see REMARKS on P. 17, Sec. of 'Claims 2 and 16')

Examiner's response:

a) As stated in the previous Office action, specifically stated in claims 1, 10, 15, and 17, Mitchell et al., which were directly applied to the rejections, not Cahill et al., clearly disclose the followings:

- "static metrics alone may be insufficient in evaluating the dynamic behavior of an application at run time, as its behavior will be influenced by the operational environment as well as the complexity of the source code" (see Sec. 1, 3rd Para.)
- "to quantify the external quality of a software product using a set of dynamic metrics calculated at run-time that evaluate the product's complexity. These metrics by themselves can provide us with useful information, and can also help to calibrate the information obtained from a static analysis" (see Sec. 2, 2nd Para.)
- "The quality of a software product will be influenced by its operational environment as well as the source code complexity, therefore it was believed that measures that assess run-time quality may aid in the analysis of software quality. Static coupling and cohesion metrics deal with the architectural aspects of a software system, whereas run-time measures necessarily deal also with the behavioral aspects of the system" (see 5th Para.)
- "It is known that a one-to-one relationship does not exist between static and dynamic metrics but our study indicated that there might be some co-relation" (see Sec. III Work Done To Date, 1st Para., 3rd Para., 4th Para., 5th Para.);

- "To determine if a quantifiable correlation exists between the information obtained from a static and dynamic analysis of a program; clearly, a static analysis is relatively independent of program behavior, whereas any run-time analysis will be fundamentally influenced by the testing strategy and test inputs" (see Sec. III Work Done To Date, 1st Para., 3rd Para., 4th Para., 5th Para.)
- "Degree of Static and Dynamic Coupling within a group of classes for the non-API classes of the programs from SPEC JVM98 benchmark suite" (see Fig. 1)
- "Figs 2-3 give a pictorial representation of the relationship between the static and dynamic cohesion data ..." (see Sec. A – Coupling Metrics; Sec. B – Cohesion Metrics)

b) Mitchell et al. are applied to claims 6 and 10 in this Office action

c) Mitchell et al. are applied to claims 2 and 16 in this Office action

Conclusion

21. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Ben C. Wang whose telephone number is 571-270-

1240. The examiner can normally be reached on Monday - Friday, 8:00 a.m. - 5:00 p.m., EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on 571-272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

TUAN DAM
SUPERVISORY PATENT EXAMINER

BCW BW

November 8, 2007